NAS Grid Benchmarks Version 1.0

Rob F. Van der Wijngaart*, Michael Frumkin
NASA Advanced Supercomputing (NAS) Division
NASA Ames Research Center, Moffett Field, CA 94035-1000
wijngaar@nas.nasa.gov, frumkin@nas.nasa.gov

NASA Technical Report NAS-02-005

July 2002

Abstract

We provide a paper-and-pencil specification of a benchmark suite for computational grids. It is based on the NAS Parallel Benchmarks (NPB) and is called the NAS Grid Benchmarks (NGB). NGB problems are presented as data flow graphs encapsulating an instance of a slightly modified NPB task in each graph node, which communicates with other nodes by sending/receiving initialization data. Like NPB, NGB specifies several different classes (problem sizes). In this report we describe classes S, W, and A, and provide verification values for each. The implementor has the freedom to choose any language, grid environment, security model, fault tolerance/error correction mechanism, etc., as long as the resulting implementation passes the verification test and reports the turnaround time of the benchmark.

1 Introduction

The NAS Parallel Benchmarks (NPB) were designed to provide an objective measure of the capabilities of hardware and software systems to solve computationally intensive computational fluid dynamics problems relevant to NASA. They are considered representative of an important segment of high performance scientific computing. At the time of NPB's inception in 1991 there were no accepted standards for programming parallel computers, and there was great diversity in hardware systems. It was deemed that any specific benchmark implementation would be unfairly biased towards a certain system configuration or programming paradigm. Hence, the first version of NPB, referred to as NPB1 [1], consisted of a paper-and-pencil specification, with virtually all aspects of the implementation left to the implementor. A reference implementation, mostly

*Employee of Computer Sciences Corporation

in Fortran, was provided for convenience, but no claims were made about algorithmic efficiency or appropriateness for any particular system.

Despite its apparent lack of concreteness, NPB1 was embraced by vendors and researchers. It served as a fruitful testing ground for programming tools and compiler optimizations. Once a certain convergence in programming paradigms was reached, MPI (Message Passing Interface) being the first generally accepted standard, NAS created the source code implementation NPB2 [2], which became the de facto yardstick for testing (parallelizing) compilers and tools.

Computational grids [5, 8] are currently in a state of development comparable to that of high performance computers at the end of the 1980s. Several prototype grid tool kits exist (e.g. Globus [6], Legion [10], CORBA [3], Sun Grid Engine [9], Condor [7]), whose relative merits are not well understood. Here we describe a new benchmark suite, the NAS Grid Benchmarks (NGB), which aims to provide an exploration tool for grids, similar to that which NPB provided for high-performance computing systems. Among others, NGB addresses one of the most salient features of grid computing, namely the ability to execute distributed, communicating processes.

The pencil-and-paper specification provided here serves as a uniform tool for testing functionality and efficiency of grid environments. Users are free to implement NGB as they see fit, provided they observe the same—fairly loose—rules laid down in the NPB1 [1] report. Specifically, NGB does not specify how to implement/select the following: authentication, security, fault tolerance, scheduling, grid environment, mapping of NGB tasks onto the computational grid. An NGB result consists of a correctly reproduced set of verification values, plus the turnaround time. Other metrics, such as aggregate resources (disk space, CPU time, memory, network bandwidth) used to complete the benchmarks, are currently considered too poorly defined to have utility outside the benchmarker's own organization. An NGB implementor may provide a more detailed report on the performance of grid components, including time to communicate data between any two benchmark tasks executed on (potentially different) platforms, and wall clock time for each task. But since NGB does not specify how many or which resources to employ, the detailed report is considered informative in nature. More information about the NGB motivation, requirements, and design is provided in [4].

2 NGB design

Like NPB, NGB is made available in several different problem sizes, traditionally called classes. An NGB problem of a particular class is specified by a data flow graph encapsulating NGB tasks (NPB problems) and communications between these tasks. Each graph contains a Report node (see Section 3) that collects verification statuses to determine correctness of the computational result. The decision to use NPB problems, specifically BT, SP, LU, MG, and FT, in the definition of NGB is motivated as follows.

- Good specifications and implementations of NPB problems already exist.

Freely downloadable versions can be found at
`http://www.nas.nasa.gov/Research/Software/swdescription.html`

- NPB is well-studied, well-understood, and widely accepted as a scientific benchmark suite.
- Solid verification procedures for NPB problems already exist.
- NPB problems require no interaction and no data files to start, in principle (but see next item).
- NPB problems produce sizeable arrays representing solutions on discretization meshes. These can be used as input for any of the other NPB problems, since each is based on structured discretization meshes covering the same physical domain. Hence, it is fairly straightforward to construct simple but plausible dependency graphs representing sets of interrelated tasks on the grid, with significant data flows (solution arrays) between them.
- The granularity of the benchmark can easily be controlled by varying the number of iterations carried out by each NPB task.
- NPB problems embody operations that can sensibly symbolize scientific computation (flow solvers: SP, BT, LU), post-processing (data smoother: MG), and visualization (spectral analysis: FT). We consider collections of such tasks suitable candidates for grid computing.
- Well-implemented portable, parallel versions of all NPB problems exist, which enables balancing the load of complicated grid tasks by assigning different amounts of computational resources to different subtasks.

In order to facilitate implementation of NGB, we require only small changes to be made to the NPB building blocks. A description of these changes is provided in this report. Whenever interpolation is required to transfer information from a certain mesh size in one NPB problem to a different mesh size for another NPB problem within the same NGB instance, we prescribe tri-linear (Lagrangian) interpolation, as defined in Section 3.2.

3 NGB Data Flow Graphs

An instance of NGB comprises a collection of slightly modified NPB problems, each defined on a fixed, rectilinear discretization mesh. Each NPB problem (BT, SP, LU, MG, or FT) is specified by class (mesh size, number of iterations), source(s) of input data, and consumer(s) of solution values. Hence, an instance of NGB is specified by a *Data Flow Graph* (DFG), see Figures 1–4. The DFG consists of nodes connected by directed arcs. It is constructed such that there is a directed path from any node to the sink node of the graph (indicated by *Report*). This is necessary to ensure that any failing node will be detected.

3.1 Corrections to NPB

It has been observed by a number of researchers that the officially released implementation of NPB and the paper-and-pencil specification contain some inaccuracies and minor errors. We list these for reference, but only correct those (items 3 and 4 below) that cause problems when implementing NGB.

1. The numerical scheme used in MG does not qualify as a multigrid method; true multigrid derives its efficiency from the fact that mesh points at coarser levels of refinement are contained in all finer grids. This can be assured if the number of mesh cells doubles in each coordinate direction with every refinement. However, in NPB's MG the number of mesh *points* doubles with every refinement.

2. The MPI implementation of MG formally exhibits a race condition in the initialization.

3. The MPI implementation of FT does not scale the reverse Discrete Fourier Transform with the size of the mesh as it should according to [1]. This causes the norm of the solution field after each invocation of FT within NGB to jump by a factor of $\prod_{i=1}^{3} n_i$, where n_i is the number of mesh points in coordinate direction i. Especially for the larger problem sizes the jump becomes too large, so we always divide the NPB FT result by $\prod_{i=1}^{3} n_i$ before transferring the (real part of) the solution to a successor DFG node, but after computing checksums in case the node performs a verification test.

4. Initialization of the flow field in SP, BT, and LU is supposed to employ transfinite interpolation of the exact solution on the boundaries of the discretization mesh. However, in neither the NPB specification [1] nor its MPI implementation does the initialization correspond to any reasonable interpolation. The initialization does not even reproduce a constant flow field if all boundary values are identical. This did not lead to problems within NPB, but NGB breaks down when BT is followed by SP, as happens in HC. The reason is that the discontinuity between boundary values and initial interior solution causes BT to generate oscillations in the computed solution near the mesh boundary. This results in attempts by SP to compute the square root of a negative number when evaluating the local speed of sound, which causes the program to fail. We remedy this by employing tri-linear interpolation (see Section 3.2 below) to compute the initial flow field for SP, BT, and LU whenever they are immediate successors of the Launch node. In this process only the values of the dependent variables at the eight corners of the cubical grid are used. True transfinite interpolation, a refinement of tri-linear interpolation, creates too smooth an initial solution, which causes premature convergence.

3.2 Filtering mesh data

All mesh-based NPB problems are defined on the three-dimensional unit cube. However, even within the same problem class (S, W, or A) there are different mesh sizes for the different benchmark tasks. Discretization points of meshes of different size generally do not coincide. In order to use the output from one NPB task as input for another, we interpolate the data tri-linearly, and subsequently take arithmetic averages of multiple inputs. These operations are carried out by the nodes in the DFG labeled "MF" (Mesh Filter). The methods used by NPB preserve numerical stability under these filter operations. Let a variable u be defined at the grid points of a mesh of extent $(1{:}nx_1, 1{:}ny_1, 1{:}nz_1)$, and let v be the interpolant of the same variable at the grid points of a mesh of extent $(1{:}nx_2, 1{:}ny_2, 1{:}nz_2)$. The value of v at point (i, j, k) is calculated as follows.

$$
\begin{aligned}
v(i,j,k) = \quad & \gamma \; [\; \beta \; (\alpha u(i_h, j_h, k_h) + (1-\alpha) u(i_l, j_h, k_h)) + \\
& \qquad (1-\beta)\,(\alpha u(i_h, j_l, k_h) + (1-\alpha) u(i_l, j_l, k_h))] \\
& + \\
& (1-\gamma)[\; \beta \; (\alpha u(i_h, j_h, k_l) + (1-\alpha) u(i_l, j_h, k_l)) + \\
& \qquad (1-\beta)\,(\alpha u(i_h, j_l, k_l) + (1-\alpha) u(i_l, j_l, k_l))]
\end{aligned}
\tag{1}
$$

where

$$
\begin{array}{lll}
dx = 1/(nx_2 - 1) & dy = 1/(ny_2 - 1) & dz = 1/(nz_2 - 1) \\
x = (i-1)dx(nx_1 - 1) & y = (j-1)dy(ny_1 - 1) & z = (k-1)dz(nz_1 - 1) \\
i_h = \min(\lfloor x + 2 \rfloor, nx_1) & j_h = \min(\lfloor y + 2 \rfloor, ny_1) & k_h = \min(\lfloor z + 2 \rfloor, nz_1) \\
i_l = i_h - 1 & j_l = j_h - 1 & k_l = k_h - 1 \\
\alpha = x - i_l & \beta = y - j_l & \gamma = z - k_l
\end{array}
\tag{2}
$$

If u and v are vector fields the interpolation is performed in a componentwise fashion.

3.3 DFG Node

Each node (except *Launch* and *Report*) represents a single computational task, which consists either of filtering or of solving one of the NPB problems. It has a set of input and output arcs, and a compute module, to be implemented as an independently running process or collection of processes/threads. If a node is connected to the source node (indicated by *Launch* in Figures 1–4), it receives control directives to initiate the computation. Otherwise it receives input data from other nodes through its input arc(s), to be used to calculate or set initial conditions. If the node is not connected to the sink node (indicated by *Report* in Figures 1–4), it sends the computed solution along all output arcs. The implementor is free to attach to a node any additional attributes, such as information on the computational resources required for performing its functions, which can be used by a scheduler. The sink node collects verification statuses of any nodes connected to it.

3.4 DFG Arc

An arc connects tail and head nodes and represents transmission of data from the tail to the head. The implementor is free to attach to the arc any additional attributes, such as information on the communication volume and frequency, which can be used by a scheduler. Data to be exchanged between DFG nodes may not be precomputed or cached, but must be created anew for each benchmark run. Dashed arcs in Figures 1–4 connect the nodes *Launch* and *Report* to the rest of the graph. They carry no computational data, but are required for control and timing. NGB does not prescribe the mechanism for transferring data, nor does it prescribe the representation of data to be exhanged between DFG nodes. This has some implications for the verification tests performed by the nodes connected to the Report node, as descibed below.

3.5 Four NGB Problems

NGB consists of families of problems named Embarrassingly Distributed (ED), Helical Chain (HC), Visualization Pipeline (VP), and Mixed Bag (MB). They are described in detail in the following subsections. Whenever two NPB tasks in the DFG exchange solution data (an array), with or without a mesh filter operation in between, the type of array to be transferred is determined by the receiving task, except in the case of FT (see below). Specifically, when SP, BT, and LU communicate with each other, they exchange the solution on the entire mesh, consisting of five double precision words per mesh point. When SP, BT, or LU send data to an MG node–which operates on data that consists of a single double precision word per mesh point–they compute and send the local speed of sound a for each mesh point. The solution at a point is defined by the vector $\mathbf{u}$, with five components (see [1]). The speed of sound is defined by:

$$a = \left(0.56(u_5 - \frac{1}{2u_1}(u_2^2 + u_3^2 + u_4^2))/u_1\right)^{1/2}. \qquad (3)$$

When MG sends data to an FT node, it transfers the solution on the entire mesh (a single double precision word per point), except the solution values on the periodic boundaries. When FT sends data to an FT node, only the real part of the complex solution value on the entire mesh is transferred. See Table 1, which lists the types of data exchanged between NPB tasks within NGB. Some of the DFG arc tail/head combinations listed in the table only occur for NGB sizes larger than A, to be specified in a later report.

Scaleup of NGB problems from class S to larger problem sizes is accomplished by increasing the number of graph nodes, as described below, as well as the size of the NPB problem in each node. Specifically, for NGB class X we employ NPB problems of class X exclusively as well.

Other than ED, each NGB problem involves transfer of solution values between DFG nodes. This is a potential source of numerical error, because nodes may execute on different architectures, with different data representations and/or arithmetic. The original NPB verification tests are fairly tight, to avoid the possibility of validating an erroneous solution. Since NGB uses the same rather strict error tolerances, it is possible that correctly computed solutions fail the verification test. Specifically, if error norms of the final NGB solution(s) have a small absolute magnitude, the build-up of errors due to differing arithmetic or to data conversions may exceed the verification threshold. For example, if the magnitude of an actual error norm is 10^{-7} and data conversion causes differences in the last bit of a double precision number with a 48-bit mantissa, an error tolerance of 10^{-7} may cause a correctly computed solution to fail the verification test. To avoid this problem, which manifests itself for class S of the NGB, we cut the size of the time step of SP, BT, and LU for that class in half (except for ED, which does not suffer from data conversions because no communication takes place). This slows down convergence sufficiently that final error norms are well above the threshold value for triggering false verification failures.

3.5.1 Embarrassingly Distributed (ED)

ED represents the important class of grid applications called parameter studies, which constitute multiple independent runs of the same program, but with different input parameters.

We select SP, the core of an important class of flow solvers, as the basis for this benchmark. There is no communication between any of the instantiations of SP, as indicated in Figure 1 depicting class S (sample size) of the benchmark. If we number the nodes of the ED graph consecutively, starting from zero, then the parameter study is defined by varying the initialization constant $C_{1,1}$, as defined in [1], as follows: $C_{1,1} = 2(1 + i * 0.01)$, where i is the node number. Note that the initialization of the flow field in the interior of the mesh takes place through tri-linear interpolation of the flow variables at the eight corner points of the mesh, see Section 3.1. No other changes are made to the NPB problem defining SP.

Table 1: Types of field data (double precision real) exchanged by NPB tasks

		Arc head				
		BT	SP	LU	MG	FT
Arc tail	BT	5-vector	5-vector	5-vector	scalar	–
	SP	5-vector	–	5-vector	–	–
	LU	5-vector	–	–	scalar	–
	MG	–	–	–	–	scalar
	FT	–	–	–	–	real part of complex scalar

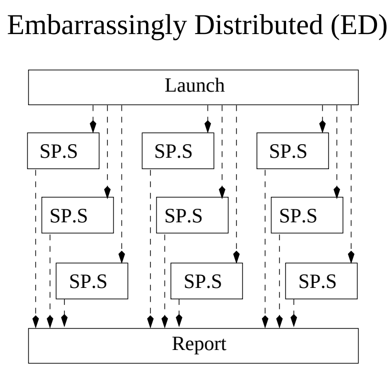

Figure 1: Data flow graph of NGB problem ED, class S (sample size). Dashed arrows signify control flow.

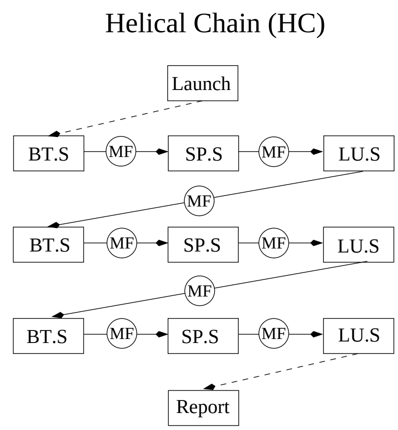

Figure 2: Data flow graph of NGB problem HC, class S (sample size). Solid arrows signify data and control flow. Dashed arrows signify control flow only.

3.5.2 Helical Chain (HC)

HC represents long chains of repeating processes, such as a set of flow computations that are executed one after the other, as is customary when breaking up long running simulations into series of tasks. We select BT, SP, and LU to make up the successive nodes in the chain, and connect this triplet into a linear chain, as shown in Figure 2. Initialization of the computation takes place in the regular NPB style for the first BT node in the graph (but see Section 3.1 for the correction we need to apply). Subsequent nodes use the final computed solution of their predecessor node to initialize. For problem classes S and A no interpolation is required, because the mesh sizes for SP, BT, and LU are identical. However, for class W they differ, so in general interpolation is required, as indicated by the MF nodes in the graph.

3.5.3 Visualization Pipeline (VP)

VP represents chains of compound processes, like those encountered when visualizing flow solutions as the simulation progresses. It comprises the three NPB problems BT, MG, and FT, which fulfill the role of flow solver, post processor, and visualization module, respectively. This triplet is linked together into a logically pipelined process, where subsequent flow solutions can be computed while postprocessing and visualization of previous solutions is still in progress. This process is illustrated in Figure 3.

Data exchange between NPB nodes in VP is as follows. Each BT node transfers its entire solution to its BT successor node in the pipeline, with no

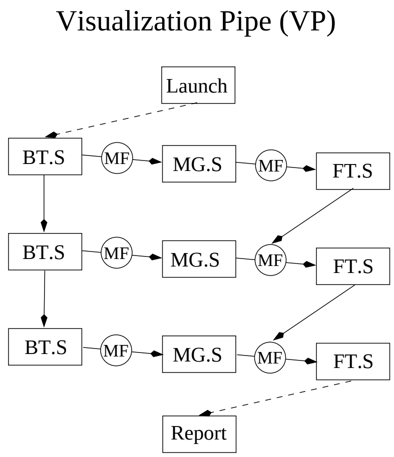

Figure 3: Data flow graph of NGB problem VP, class S (sample size). Solid arrows signify data and control flow. Dashed arrows signify control flow only.

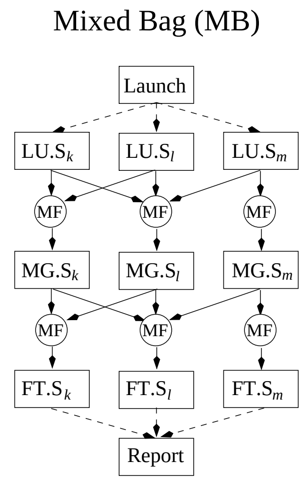

Figure 4: Data flow graph of NGB problem MB, class S (sample size). Solid arrows signify data and control flow. Dashed arrows signify control flow only. Subscripts indicate different (relative) numbers of iterations.

interpolation necessary. Initialization of the computation takes place in the regular NPB style for the first BT node in the graph (but see Section 3.1 for the correction we need to apply). Each BT node also produces the scalar field of sound speeds, consisting of one double precision number for each point in the BT mesh. This solution is used to initialize the MG successor node, but only in the interior of the mesh. Hence, the BT sound speed field is interpolated (by the MF nodes in the graph) onto the interior of the MG mesh. This interior, which is discretized by a mesh whose numbers of points are exact powers of two in each coordinate direction, covers the unit cube, and thus coincides with the BT mesh in physical space. The boundary values for MG are set by using explicit periodic boundary conditions on all six faces of the cubic mesh, which means copying function values at discretization points one cell away from the mesh boundary to the corresponding boundary location on the other side of the mesh (this copying process is identical to that in the original NPB MG problem within each iteration). Similarly, upon completion MG transfers the interior values of its computed solution to the FT node via the MF node. However, for each FT node other than the first in the visualization pipeline there is also an FT solution that is used to initialize the successor FT node. As mentioned above in the description of the NGB graph set, FT uses for its initialization an array of double precision real values, consisting of the arithmetic average of the real part of the previous FT solution (if present) and the interior solution of the MG node, both interpolated onto the whole FT mesh. The scalar double precision complex initial field of FT is created out of the real part as follows. For mesh point (i, j, k) of the FT mesh (the origin of the mesh is point $(1,1,1)$) the imaginary part of the initial solution u_1 is:

$$Im(u_1) = (((i+j+k) \bmod 3) - 1) * Re(u_1)\,. \tag{4}$$

The reason FT nodes communicate with each other is that it must be possible to detect whether each node has finished computations successfully.

3.5.4 Mixed Bag (MB)

MB is similar to VP. It again involves the sequence of flow computation, post-processing, and visualization, but now the emphasis is on introducing asymmetry. Different amounts of data are transferred between different tasks, and some tasks require more work than others, making it hard for a grid scheduler to map DFG tasks to grid resources efficiently. The MB DFG specifies that several flow computations can start simultaneously (the first horizontal layer), but the next layer implies some synchronization when computed solutions from multiple instantiations of LU are used as—interpolated, averaged—input for the MG nodes, see Figure 4. The same communication/synchronization pattern is used for transfer of data between MG and FT nodes. As in the case of VP, the interior of the MG mesh is initialized with the sound speed field from the entire LU mesh, and MG also transfers —interpolated, averaged—solution values on the interior of its mesh to the entire mesh of FT successor nodes. Also, as in the case of VP, the double precision complex initial field of FT is constructed from double precision real data using Equation 4. An additional complexity of MB is that nodes in different columns of the DFG perform different (relative) numbers of time steps/iterations, indicated by the subscripts k, l, and m in Figure 4. This creates a potential for load imbalance that must be handled by allocating different amounts of resources to computational nodes in the same layer of the graph. The mechanism for determining the relative number of time steps for the graph nodes is as follows. Let the number of iterations for a node in the leftmost column in graph layer d be N_d^0, which is determined by dividing the number of iterations of the original NPB by the *depth* of the DFG, with a minimum of one, as described in Section 3.6. Also, let the column index of a graph node be c, starting with zero for the leftmost column, and increasing with unit steps when moving to the right[1]. Then the number of iterations for any node is defined by $N_d^c = \max(1, \lfloor N_d^0(1 - \frac{c}{2(c+1)})\rfloor)$. For class S the numbers of iterations executed by LU nodes of increasing column index are 16, 12, and 10, by MG nodes 1, 1, and 1, and by FT nodes 2, 1, and 1, respectively. Initialization of the computations in the first graph layer is the same as in the regular NPB (but see Section 3.1 for the correction we need to apply).

[1] For class S the MF node preceding the FT node with column index zero has two input arcs, that with column index one has three input arcs, and that with column index two has one input arc.

3.6 Scaling Rationale and Verification

The purpose of NGB is to gauge the capabilities of computational grids. It is expected that as time progresses, such grids will become more powerful both in terms of single system performance, as well as in terms of the number of accessible systems. The creation of larger NGB problem sizes (classes) is meant to capture this expected development. Hence, successive NGB classes will involve more computational work, as well as more graph nodes, in general. The rationale for selecting the parameters that determine these is as follows.

The NPB classes already capture growth in problem size. NGB makes use of this by employing for each class a data set (grid or array) of the same size as the corresponding NPB class. Furthermore, we accept the premise that an NGB instance of a certain class should run approximately as long as an instance of NPB of the same class if we disregard communication times between graph nodes, interpolation and averaging of input data, and exploitation of intra-node parallelism. In other words, we associate with each graph node containing an NPB problem a weight equal to the amount of computational work, and make sure that the critical path has approximately the same aggregate weight as an NPB instance of that class. Other than ED, each NGB problem will have several different NPB problems on its critical path. The goal of keeping the summed weights of the nodes on the critical path the same as that of a single NPB problem may be reached by setting the number of iterations or time steps within each NGB NPB problem equal to that of the original NPB problem, divided by the number of nodes on the critical path, rounded down, with a minimum of one. Consequently, the computational work on the NGB critical path will be no more than that of the most computationally intensive NPB problem of the same class.

Note: we define the *width* of each NGB DFG as the maximum number of NPB nodes in the DFG that can be executed concurrently, and the *depth* as the length of the critical path of such nodes, ignoring the pipeline fill or drain nodes in VP. Using these definitions, the total size of each NGB DFG, not including the Report, Launch, and filter nodes, is the product of depth and width. For convenience we use the depth instead of critical path length for scaling the number of iterations or time steps within the DFGs. DFG width has no influence on turnaround time—save pipeline fill/drain time—if the NGB implementor fully exploits inter-node parallelism.

If we map consecutive classes S, W, A, ..., to consecutive integers, starting with one, we specify the depth D of NGB class i as follows.

$D^i_{ED} = 1, D^i_{HC} = 9\max(1, i-2), D^i_{VP} = 3\max(1, i-2), D^i_{MB} = \max(3, i)$.

The corresponding width W is defined as follows.

$W^i_{ED} = 9 * 2^{\max(0,i-3)}, W^i_{HC} = 1, W^i_{VP} = 3, W^i_{MB} = \max(3, i)$.

We list the numerical values of the total numbers of nodes for benchmark classes S, W, and A through E, respectively, in Table 2. While it is clear which NPB problems to select for the different graph layers of ED, HC, and VP, it is not obvious for MB. We adopt the following strategy. The last graph layer (connected to the Report node) is assumed to have depth zero. Depth increases by one for each higher layer. Layers at depths 0 and 1 always employ FT and MG, respectively. Layers at larger depth are assigned NPB problems LU, BT, and SP, respectively, in a cyclical fashion.

Table 2: Width $\times$ depth of NGB graphs

	Class						
Name	S	W	A	B	C	D	E
ED	9×1	9×1	9×1	18×1	36×1	72×1	144×1
HC	1×9	1×9	1×9	1×18	1×27	1×36	1×45
VP	3×3	3×3	3×3	3×6	3×9	3×12	3×15
MB	3×3	3×3	3×3	4×4	5×5	6×6	7×7

Verification values for NGB classes S, W, and A are presented in the Appendix. For ED these values are required for each node in the graph, for HC and VP only for the last node, and for MB for all nodes in the last layer of the graph. Verification procedures for ED and HC are the same as those for the original NPB, although the actual values differ, of course. For VP and MB verification values and procedures must be defined for FT nodes, since these are connected to the report node. In the original NPB each step in the reverse FFT computes a double precision complex checksum and compares it with a verification value. The reason for this is that the reverse FFT steps are independent, and verifying only the last is not sufficient. In NGB the number of verification values is potentially significantly larger than in NPB. To keep the number within reason, we verify not every reverse FFT step individually, but compute the arithmetic average of the checksums over all the reverse FFT steps within each FT task and compare that to a verification value.

4 Summary and Future Work

We have described the composition and details of a suite of four families of grid benchmark problems called NAS Grid Benchmarks (NGB), based on the NAS Parallel Benchmarks (NPB). Only classes S, W, and A, based on NPB classes S, W, and A, are specified. We provide verification values for these classes, so that correctness of the computed solutions can be determined. Verification values for larger classes will be presented in a future report.

References

[1] D. Bailey, E. Barscz, J. Barton, D. Browning. R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, S. Weeratunga. *The NAS Parallel Benchmarks.* NAS Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, CA, 1994.

[2] D.H. Bailey, T. Harris, W.C. Saphir, R.F. Van der Wijngaart, A.C. Woo, M. Yarrow. *The NAS Parallel Benchmarks 2.0.* NAS Technical Report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, 1995.

[3] R. Ben-Naten. *CORBA: A Guide to Common Object Request Broker Architecture.* McGraw-Hill, New York, 1995.

[4] M. Frumkin, R.F. Van der Wijngaart, *NAS Grid Benchmarks: A Tool for Grid Space Exploration,* Proc. 10^{th} Int'l Symp. High Performance and Distributed Computing conference, San Francisco, August 2001.

[5] *The Grid. Blueprint for a New Computing Infrastructure.* I. Foster, C. Kesselman, Eds., Morgan Kaufmann Publishers Inc., San Francisco, CA, 1999.

[6] I. Foster, C. Kesselman. *Globus: A Metacomputing Infrastructure Toolkit.* Int. J. Supercomputer Applications, 11(2):115-128, 1997, http://www.globus.org.

[7] M. Livny, J. Basney, R. Raman, T. Tannenbaum. *Mechanisms for High Throughput Computing.* SPEEDUP Journal, Vol. 11(1), 1997, http://www.cs.wisc.edu/condor/.

[8] *NASA Information Power Grid.* http://www.nas.nasa.gov/IPG.

[9] *Codine 5.2 Manual, Revision A.* Sun Microsystems, Inc., Palo Alto, CA, September 2000, http://www.sun.com/gridware.

[10] *Legion 1.6, Developer Manual.* The Legion Research Group, Dept. Computer Science, U. Virginia, Charlottesville, VA, 1999, http://legion.virginia.edu.

5 Appendix: Verification values

The NGBs require that the correct solutions be computed. This is ensured by demanding that verification values for solution and error norms of all nodes that are directly connected to the Report node in the DFG are computed. In this section we provide these values. Error tolerances and methods for computing norms for NGB are the same as for NPB.

5.1 Embarrassingly Distributed

Verification values for ED, class S, computed by each SP node

node #	solution norm	error norm
	0.8676543011741d-04	0.8719352691803d-07
	0.6033493408310d-04	0.5988808458188d-07
0	0.6278114032528d-04	0.6270092156558d-07
	0.6603179902638d-04	0.6551568383242d-07
	0.9122028866543d-04	0.8553969317102d-07
	0.8601620225691d-04	0.8643202500288d-07
	0.5988773196543d-04	0.5943418706437d-07
1	0.6157026954073d-04	0.6155109448566d-07
	0.6457044508652d-04	0.6413646078612d-07
	0.8687428401194d-04	0.8152493003792d-07
	0.8528517607623d-04	0.8568916795314d-07
	0.5944509868286d-04	0.5898555351009d-07
2	0.6043095623721d-04	0.6046719704763d-07
	0.6319041663855d-04	0.6283245382631d-07
	0.8273663135464d-04	0.7770753603531d-07
	0.8457166533017d-04	0.8496424291102d-07
	0.5900696189910d-04	0.5854206620123d-07
3	0.5935880462626d-04	0.5944517278358d-07
	0.6188715115064d-04	0.6159941928907d-07
	0.7879773267341d-04	0.7407863788365d-07
	0.8387501686770d-04	0.8425657226531d-07
	0.5857325969939d-04	0.5810362059449d-07
4	0.5834963218012d-04	0.5848116700380d-07
	0.6065629301996d-04	0.6043330938920d-07
	0.7504851869600d-04	0.7062984804457d-07
	0.8319460928597d-04	0.8356551209019d-07
	0.5814393866091d-04	0.5767012327603d-07
5	0.5739946515836d-04	0.5757152229935d-07
	0.5949368685757d-04	0.5933026612155d-07
	0.7148042299123d-04	0.6735324051262d-07

node #	solution norm	error norm
	0.8252985024299d-04	0.8289044920091d-07
	0.5771895249772d-04	0.5724149042257d-07
6	0.5650453308785d-04	0.5671277285472d-07
	0.5839537229112d-04	0.5828661632209d-07
	0.6808535665169d-04	0.6424132668705d-07
	0.8188017551638d-04	0.8223080020494d-07
	0.5729826099434d-04	0.5681764661514d-07
7	0.5566126383428d-04	0.5590163933420d-07
	0.5735757816226d-04	0.5729886620119d-07
	0.6485568447004d-04	0.6128703228573d-07
	0.8124504686581d-04	0.8158600910848d-07
	0.5688182884348d-04	0.5639852350520d-07
8	0.5486627790428d-04	0.5513502305850d-07
	0.5637671694675d-04	0.5636369592773d-07
	0.6178420280687d-04	0.5848367565086d-07

Verification values for ED, class W

node #	solution norm	error norm
	0.1745133059397d-04	0.6955341579879d-06
	0.1194347497651d-04	0.4732433767510d-06
0	0.1271275032480d-04	0.5071720177863d-06
	0.1150480909799d-04	0.4600674574080d-06
	0.1305114022206d-04	0.5381322068507d-06
	0.1745560352290d-04	0.6957392189784d-06
	0.1193739401471d-04	0.4730069614536d-06
1	0.1269942115908d-04	0.5066479195335d-06
	0.1149066787325d-04	0.4595086246319d-06
	0.1301167555462d-04	0.5365356761184d-06
	0.1745986923555d-04	0.6959438483249d-06
	0.1193141412701d-04	0.4727745781123d-06
2	0.1268621099471d-04	0.5061284384576d-06
	0.1147665934807d-04	0.4589549875939d-06
	0.1297237680057d-04	0.5349455062024d-06
	0.1746412767819d-04	0.6961480441358d-06
	0.1192553395452d-04	0.4725461723920d-06
3	0.1267311889347d-04	0.5056135396850d-06
	0.1146278217756d-04	0.4584064944105d-06
	0.1293324425385d-04	0.5333617144459d-06
	0.1746837879467d-04	0.6963518044347d-06
	0.1191975215752d-04	0.4723216907465d-06
4	0.1266014391456d-04	0.5051031881884d-06
	0.1144903502624d-04	0.4578630935544d-06
	0.1289427815335d-04	0.5317843158727d-06

node #	solution norm	error norm
	0.1747262252970d-04	0.6965551273059d-06
	0.1191406741612d-04	0.4721010804231d-06
5	0.1264728511460d-04	0.5045973487811d-06
	0.1143541657009d-04	0.4573247339124d-06
	0.1285547868348d-04	0.5302133232448d-06
	0.1747685882701d-04	0.6967580108191d-06
	0.1190847842890d-04	0.4718842894206d-06
6	0.1263454154958d-04	0.5040959861974d-06
	0.1142192549392d-04	0.4567913647034d-06
	0.1281684597929d-04	0.5286487472472d-06
	0.1748108763020d-04	0.6969604530568d-06
	0.1190298391404d-04	0.4716712665253d-06
7	0.1262191227311d-04	0.5035990650298d-06
	0.1140856049497d-04	0.4562629356041d-06
	0.1277838012802d-04	0.5270905965546d-06
	0.1748530888245d-04	0.6971624520995d-06
	0.1189758260757d-04	0.4714619612488d-06
8	0.1260939633998d-04	0.5031065498690d-06
	0.1139532028062d-04	0.4557393966743d-06
	0.1274008117128d-04	0.5255388779393d-06

Verification values for ED, class A

node #	solution norm	error norm
	0.1662388872593d-03	0.6779616046334d-06
	0.1120336284839d-03	0.4612029937003d-06
0	0.1155494269554d-03	0.4944689998311d-06
	0.1144591065562d-03	0.4485901415827d-06
	0.1164120101231d-03	0.5246454435912d-06
	0.1643876342654d-03	0.6781623235228d-06
	0.1107586934055d-03	0.4609720478128d-06
1	0.1139204792057d-03	0.4939569076487d-06
	0.1126757070066d-03	0.4480440569950d-06
	0.1116344021422d-03	0.5230883044555d-06
	0.1625918474544d-03	0.6783626225632d-06
	0.1095197898803d-03	0.4607450429708d-06
2	0.1123587585230d-03	0.4934493367655d-06
	0.1109676929314d-03	0.4475030647124d-06
	0.1071044513952d-03	0.5215374339158d-06
	0.1608490693651d-03	0.6785624992312d-06
	0.1083153581826d-03	0.4605219257276d-06
3	0.1108604672866d-03	0.4929462520671d-06
	0.1093308651249d-03	0.4469671125838d-06
	0.1028086486843d-03	0.5199928417464d-06

node #	solution norm	error norm
	0.1591569849646d-03	0.6787619510589d-06
	0.1071439334321d-03	0.4603026434516d-06
4	0.1094220422343d-03	0.4924476183695d-06
	0.1077612746219d-03	0.4464361489565d-06
	0.9873431034114d-04	0.5184545362024d-06
	0.6789609756324d-06	0.6789609756324d-06
	0.4600871443050d-06	0.4600871443050d-06
5	0.4919534004407d-06	0.4919534004407d-06
	0.4459101226618d-06	0.4459101226618d-06
	0.5169225240005d-06	0.5169225240005d-06
	0.1559163057636d-03	0.6791595706091d-06
	0.1048946693312d-03	0.4598753772286d-06
6	0.1067116281816d-03	0.4914635630149d-06
	0.1048091731788d-03	0.4453889830283d-06
	0.9120310319412d-04	0.5153968104229d-06
	0.1543637155168d-03	0.6793577337061d-06
	0.1038143074591d-03	0.4596672919413d-06
7	0.1054335631917d-03	0.4909780708165d-06
	0.1034198910287d-03	0.4448726798508d-06
	0.8772452336450d-04	0.5138773993154d-06
	0.1528538118743d-03	0.6795554626864d-06
	0.1027618953918d-03	0.4594628389051d-06
8	0.1042031878741d-03	0.4904968885790d-06
	0.1020842773640d-03	0.4443611634404d-06
	0.8442389321125d-04	0.5123642931624d-06

5.2 Helical Chain

Verification values for HC, computed by final LU node

class	solution norm	error norm	surface integral
S	0.5218111814670d-03	0.2089313120425d-04	0.7840627336810d+01
	0.3707865163709d-03	0.1476501931249d-04	
	0.3800195018967d-03	0.1525298768678d-04	
	0.3397401925927d-03	0.1366601055353d-04	
	0.2759004448851d-03	0.1177099812324d-04	
W	0.2044902312107d-01	0.1321514192265d-01	0.1116573970136d+02
	0.1588666348974d-01	0.9789942432742d-02	
	0.1504465470195d-01	0.8730718331532d-02	
	0.1309985413018d-01	0.7382890378185d-02	
	0.8059897367087d-02	0.2473115882853d-02	
A	0.4568545705333d-01	0.1826624824076d-02	0.1208035142313d+02
	0.3363154291261d-01	0.1336130626708d-02	
	0.3229649387852d-01	0.1297209081421d-02	
	0.2835838318897d-01	0.1142645851712d-02	
	0.2033297513577d-01	0.8878915256529d-03	

5.3 Visualization Pipeline

Verification values for VP, computed by final FT node

class	Re(checksum)	Im(checksum)
S	-8.994899992758d+04	-3.251690423310d+04
W	-4.655638928393d+06	-5.590164304487d+04
A	-5.741701238090d+06	-3.237222308450d+04

5.4 Mixed Bag

Verification values for MB, computed by FT nodes in final graph layer

class	col.index	Re(checksum)	Im(checksum)	# iterations
S	0	-8.977825791271d+04	-3.245251953627d+04	2
	1	-8.963654068307d+04	-3.324636067181d+04	1
	2	-8.935307569342d+04	-3.313613239670d+04	1
W	0	-3.892598836945d+06	-4.672989296527d+04	2
	1	-3.529619667886d+06	-4.606914807530d+04	1
	2	-2.803661281496d+06	-3.656395484244d+04	1
A	0	-5.371673190742d+06	-3.047905369011d+04	2
	1	-5.352454993544d+06	-3.573635637856d+04	1
	2	-5.314018425421d+06	-3.554626087459d+04	1